ELSEVIER

Discrete Applied Mathematics 65 (1996) 319–345

DISCRETE APPLIED MATHEMATICS

An improved tabu search approach for solving the job shop scheduling problem with tooling constraints

Alain Hertz[a,*], Marino Widmer[b]

[a]*Département de Mathématiques, Ecole Polytechnique Fédérale de Lausanne, Chaire de Recherche Opérationnelle, CH-1015 Lausanne, Switzerland*

[b]*Université de Fribourg, IIUF, CH-1700 Fribourg, Switzerland*

Received 15 February 1993; revised 1 April 1994

Abstract

Flexible manufacturing systems (FMSs) are nowadays installed in the mechanical industry. In such systems, many different part types are produced simultaneously and it is necessary to take tooling constraints into account for finding an optimal schedule.

A heuristic method is presented for solving the m-machine, n-job shop scheduling problem with tooling constraints. This method, named TOMATO, is based on an adaptation of tabu search techniques and is an improvement on the JEST algorithm proposed by Widmer in 1991.

Keywords: Flexible manufacturing system; Job shop scheduling; Tooling constraints; Tabu search

1. Introduction

Production systems, especially in mechanical industry, are in constant evolution. Rigid transfer lines are nowadays progressively replaced by flexible cells or flexible manufacturing systems. Such systems are composed of numerically controlled machines (NC machines) which can process different kinds of parts. This versatility is essential in the current economical context: indeed, it is important to be able to produce quickly different part types to satisfy the customer requests. This means that the machine set-up times between the production of two different part types have to be as short as possible. To reach this objective, optimal tool management is necessary.

This fact makes obsolete one of the main assumptions of the job shop scheduling problem [2]: *the set-up times for the operations are sequence independent and are included in the processing times.*

*Corresponding author.

0166-218X/96/$15.00
SSDI 0166-218X(95)00040-2

Let us define more precisely the job shop scheduling problem with tooling constraints: the main elements are a set of m machines and a collection of n parts to be produced on these machines. With each part type is associated a specific process plan, which consists in a sequence of operations. Each operation is defined by a machine type on which it must be performed, its processing time and the tools which are needed. The main assumptions are the following [2, 31]:

- a machine can process only one part at a time;
- the m machines are continuously available (no breakdown and no maintenance operation are considered);
- each machine has a tool magazine with a limited capacity;
- a set of n multiple operation parts is available for processing at time zero;
- a part can be processed by only one machine at a time;
- the process plans are known in advance;
- no individual operation can be pre-empted;
- each individual operation needs a number of tools, which never exceeds the capacity of the tool magazine.

The objective is to minimize the *makespan* (which is defined as the maximum of the completion times of all operations).

Since the tool magazines have a limited capacity, tool changes have to be planned. When a tool change occurs on a machine, some tools of the magazine are replaced by others which are required for the next parts of the sequence on this machine. The time needed to change tools on a machine is usually not negligible.

It often happens that a tool may be used for producing different part types; in such a case this tool need not be duplicated in the tool magazine and space is saved in it. For example, let A, B and C be three parts which have to be processed (in this order) on a machine. Let us assume that the capacity of the tool magazine is equal to 10 and that parts A, B and C require, respectively, 4, 5 and 3 tools. If all tools needed by A, B and C are different, then a tool change must take place after the completion of A or B. However, if A and B need two common tools, no tool change is necessary for processing A, B and C since these three parts need $4 + 5 + 3 - 2 = 10$ different tools.

Let $\mathfrak{O} = \{1, \ldots, N\}$ be the total set of operations which are to be performed. For each operation $i \in \mathfrak{O}$ we denote:

- P_i the part to which i belongs;
- M_i the machine on which i is processed;
- d_i the processing time of i;
- $PP(i)$ the operation which precedes i in the process plan of P_i;
- $SP(i)$ the operation which follows i in the process plan of P_i;
- $\mathscr{T}(i)$ the set of tools needed to process i.

The capacity of the tool magazine of a machine k ($1 \leqslant k \leqslant m$) is denoted μ_k.

The job shop scheduling problem without tool management will be called JP while JPT will denote the job shop scheduling problem with tooling constraints.

Some authors have proposed models integrating tool management on a single machine. Heuristic and exact methods for minimizing either the number of tool

switches or the number of switching instants on one machine have been proposed by Tang and Denardo [28]. Bard [3] has developed a non-linear integer programming model to minimize the number of tool switches on one machine. Recently, Crama [6] and Follonier [12] have proposed efficient heuristic methods for the same problem.

Tooling constraints on several machines have also been studied by some authors: Berrada and Stecke [5] model the tool-loading problem as a non-linear program; Proust [25] solves the flow shop scheduling problem with set-up, processing and removal times; tool requirements in multi-level machining systems are studied by Koulamas [20]; de Werra and Widmer [9] suggest an integer programming model for the tool-loading problem on several machines with one-operation jobs; Mottet and Widmer [22] have developed a heuristic method for minimizing the idle time on each machine while having as few tool cassettes (small capacity magazines) in the system; a tabu search approach have been proposed by Widmer [31] for solving the job shop scheduling problem with tooling constraints; Follonier [11] tries to group parts and tools for minimizing the tool changes among identical machines.

Finally, let us mention the paper of Veeramani et al. [30] which identifies critical areas of research for the development of tool management systems in CIM.

For finding an optimal solution to a combinatorial optimization problem, various techniques have continuously been proposed in the literature. Tabu search is an iterative solution method for complex combinatorial optimization problems which has been successfully adapted to a large collection of applications [18], particularly to production problems [4], including the flow shop and the job shop problems [7, 23, 27, 32].

Widmer [31] has applied tabu search to the job shop problem with tooling constraints. We present in this paper a new adaptation of tabu search for the JPT. Numerical tests on benchmark problems indicate that the proposed algorithm outperforms the heuristic method in [31].

The basic ingredients of tabu search are described in Section 2 and adaptations of this technique to the job shop problem (with or without tooling constraints) are discussed in Section 3. The new algorithm TOMATO is described in Section 4 and computational results are contained in Section 5. Final remarks and conclusions are given in Section 6.

2. Tabu search techniques

Tabu search is a metaheuristic designed for solving combinatorial optimization problems. It has been first suggested by Glover [13] and independently by Hansen [17] for a specific application, and later developed in a more general framework [8, 14–16, 19]. We shall sketch here the basic ideas of the technique.

Assume that an objective function f has to be minimized on a set X of feasible solutions. A neighborhood $N(s)$ is defined for each solution s in X. The set X and the

definition of the neighborhood $N(s)$ induce a state space graph $\mathscr{G}$ (which may by the way be infinite): the vertex set is X and there is an arc linking a vertex s to a vertex s' if $s' \in N(s)$. Tabu search is basically an iterative procedure which starts from an initial feasible solution and tries to reach an optimal solution by moving step by step in the state space graph $\mathscr{G}$. Each step consists in first generating a collection $V^*(s)$ of solutions in the neighborhood $N(s)$ of a current solution s, and then moving to the best solution s' in $V^*(s)$, according to a function g which may be different from f. Hence

$$s' = \underset{x \in V^*(s)}{\operatorname{argmin}} \{g(x)\}.$$

Note that we may have $f(s') > f(s)$. The solutions consecutively visited in the iterative process induce an oriented path in $\mathscr{G}$.

The set X of feasible solutions and the neighborhood $N(s)$ should be defined in such a way that they induce a state space graph $\mathscr{G}$ having the following property:

> Given any feasible solution s of X, there exists an oriented path in $\mathscr{G}$ linking s to at least one optimal solution. (Property $\mathfrak{P}$)

A risk of cycling exists as soon as a solution s' no better than s is accepted. In order to prevent cycling to some extent, modifications which would bring us back to a previously visited solution should be forbidden. But it may sometimes be useful to come back to an already visited solution and continue the search in another direction from there. This is realized by keeping a list T containing the last k modifications (k may be fixed or variable). Whenever a modification m is made for moving from s to s', m is introduced in T and its reverse is considered as tabu. Since the tabu list T is finite, a cycle may happen, preventing the procedure from reaching a global optimum; hence $\mathfrak{P}$ is a necessary but not sufficient condition for ensuring that tabu search will always reach an optimal solution starting from any initial feasible one.

Finding the best solution in $V^*(s)$ may sometimes be a non-trivial matter. It may be necessary to solve the optimization problem $\min\{g(x) \mid x \in V^*(s)\}$ by a heuristic procedure. More details on the tabu search metaheuristic are given in recent overviews [16, 19].

3. Tabu search for the job shop problem

3.1. Tabu search for the JP

The job shop scheduling problem can be represented as a graph theoretical problem [26]. Given an instance of the job shop problem, the following graph $G = (V, A, E)$ can be associated with it:

- $V = \{0, \ldots, N+1\} = \mathfrak{O} \cup \{0, N+1\}$, where 0 and $N+1$ are special vertices which identify the start and the completion of the schedule;
- $A = \{(i,j) \mid 1 \leqslant i,j \leqslant N \text{ and } j = SP(i)\}$
  $\cup \{(0,i) \mid 1 \leqslant i \leqslant N \text{ and } i \text{ is the initial operation on part type } P_i\}$
  $\cup \{(i,N+1) \mid 1 \leqslant i \leqslant N \text{ and } i \text{ is the final operation on part type } P_i\}$;
- $E = \{[i,j] \mid 1 \leqslant i,j \leqslant N, M_i = M_j \text{ and } P_i \neq P_j\}$.

The length of an arc $(i,j) \in A$ or of an edge $[i,j] \in E$ with $i \geqslant 1$ is the duration d_i of operation i. All arcs issued from vertex 0 are of length zero. In other words, the set A of arcs represents the different process plans. An orientation of the edges of E is called *feasible* if it does not create any circuit in G. A feasible orientation of the edges of E corresponds to a *feasible ordering* of the operations on the machines. The job shop scheduling problem consists in finding a feasible orientation of the edges of E in such a way that the longest path from 0 to $N+1$ is minimized. Two operations which need the same machine are defined *adjacent* if the completion time of the first one is equal to the starting time of the second one. An operation is said to be *critical* if it belongs to a longest path from 0 to $N+1$. A *block* is a maximal sequence of adjacent operations to be processed on the same machine and belonging to a longest path from 0 to $N+1$.

The first adaptation of tabu search to the JP has been developed by Taillard [27]. The set X of feasible solutions is defined as the set of feasible orientations. Given a feasible solution s in X, a neighbor solution $s' \in N(s)$ is obtained by permuting two consecutive critical operations that use the same machine. When two consecutive critical operations i and j are permuted, the operation j is introduced in the tabu list T: it is forbidden to permute j with the next operation on machine M_j during $|T|$ iterations.

The neighborhood structure $N(s)$ used by Taillard has the following properties which have been proved by van Laarhoven et al. [29]:

- if a solution s is feasible, then all solutions in $N(s)$ are also feasible;
- the state space graph $\mathscr{G}$ induced by X and $N(s)$ satisfies property $\mathfrak{P}$.

A second adaptation of tabu search to the JP has been proposed by Dell'Amico and Trubian [7]. Their method seems to dominate Taillard's approach. The main difference concerns the definition of the neighborhood $N(s)$: for each operation i in a block, they consider as neighbors of s those solutions that can be generated by moving i in the first or in the last position of the block to which it belongs if the corresponding solution is feasible; otherwise, operation i is moved in the position inside the block closest to the first or the last one and such that feasibility is preserved. Dell'Amico and Trubian [7] have proved that such a neighborhood induces a state space graph $\mathscr{G}$ having property $\mathfrak{P}$.

Recently, Nowicki and Smutnicki [23] have developed a third adaptation of tabu search to the JP which is based, once again, on a different definition of the neighborhood $N(s)$. For more details, the reader is referred to [23].

These three adaptations of tabu search to the JP do not take the tooling constraints into account.

3.2. Introducing tooling constraints

As mentioned in the introduction, the time needed to change tools on a machine is usually not negligible. The moment when the contents of a tool magazine is modified will be called a *switching instant*; we shall assume that it lasts $\alpha + \beta r$ time units, where α is a fixed time due to the removal of the tool magazine from the machine, β is a fixed time for each tool replacement and r is the total number of tools which must be replaced by others.

It is important to notice that an optimal ordering of the operations on the machines for the JP does not necessarily induce an optimal schedule for the JPT. Two orderings O_1 and O_2 are represented in Fig. 1: O_1 is optimal for the JP but not for the JPT while O_2 is optimal for the JPT but not for the JP.

Given an ordering of the operations, minimizing the makespan is not necessarily achieved by minimizing the number of switching instants. For example, the schedule of Fig. 2(a) has only one switching instant but requires 7 time units. This schedule does not take advantage of the fact that machine 2 is not active during one time unit after the completion of part A. By scheduling a switching instant during this idle time, one gets a schedule with two switching instants which lasts only 6 time units.

As already mentioned, given an ordering of the operations, a schedule of minimum length for the JP can easily be obtained by finding a longest path in a graph; this is a polynomially solvable problem (see for example the adaptation of Bellman's algorithm described in [27] or the $O(N)$ algorithm developed by Adams et al. [1]). For the JPT, the complexity of this problem is still open. In other words, given the process plans of the part types and an ordering of the operations on the machines, we do not know how to deal with the tooling constraints for minimizing the makespan. This problem does not appear to be much easier when the number of switching instants on each machine is given since, as illustrated in Fig. 3, it is not always optimal to schedule the switching instants either as early, or as late as possible.

However, if the process plans, an ordering of the operations and the contents of the tool magazines are given, a schedule of minimal length can also be found by solving a longest path problem: in this case, the switching instants are considered as additional operations whose duration is known, and the graph is constructed in the same way as described in Section 3.1 for the JP.

In order to take the tooling constraints into account, let us modify the definition of adjacent operations as follows: two operations which need the same machine are called *adjacent* if there is no waiting time (but possibly a switching instant) between the completion time of the first operation and the starting time of the second one.

For the JP, we have already mentioned that Taillard's and Dell'Amico and Trubian's neighborhood structures induce a state space graph $\mathscr{G}$ having property $\mathfrak{P}$. This is not the case for the JPT. Indeed, let us assume that given an ordering of the operations, we use an exact method for dealing with the tooling constraints in order to minimize the makespan. By using Taillard's or Dell'Amico and Trubian's neighborhood structure for the problem of Fig. 4, we will cycle between solutions s_1 and

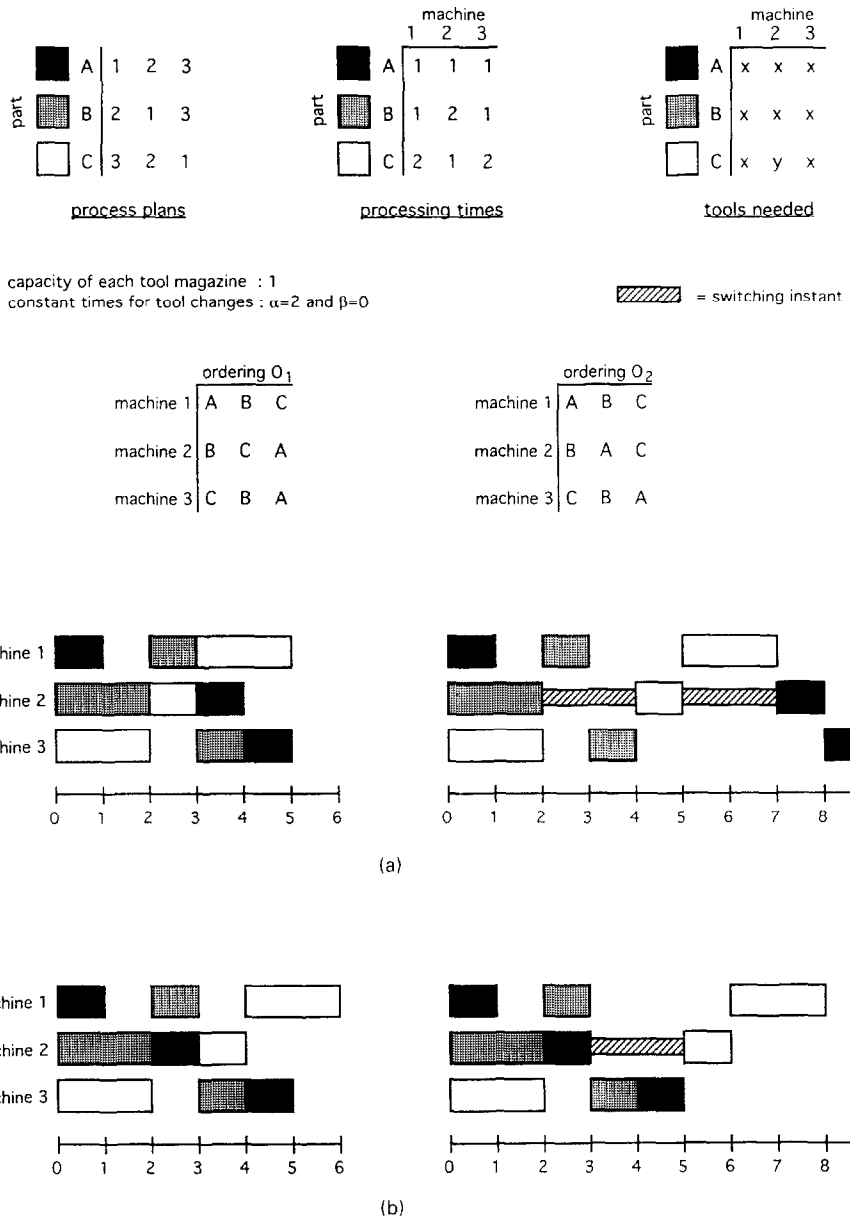


Fig. 1. (a) Best schedules induced by O_1 for the JP and JPT. (b) Best schedules induced by O_2 for the JP and JPT.

s_2 while s_3 is optimal. The solution s_3 can be obtained from s_1 by permuting two non-consecutive operations (the processing of parts A and C on machine 2). The solution s_3 can also be obtained from s_2 in two steps by permuting only consecutive operations on machine 2: first permute the non-critical operation on part C with the

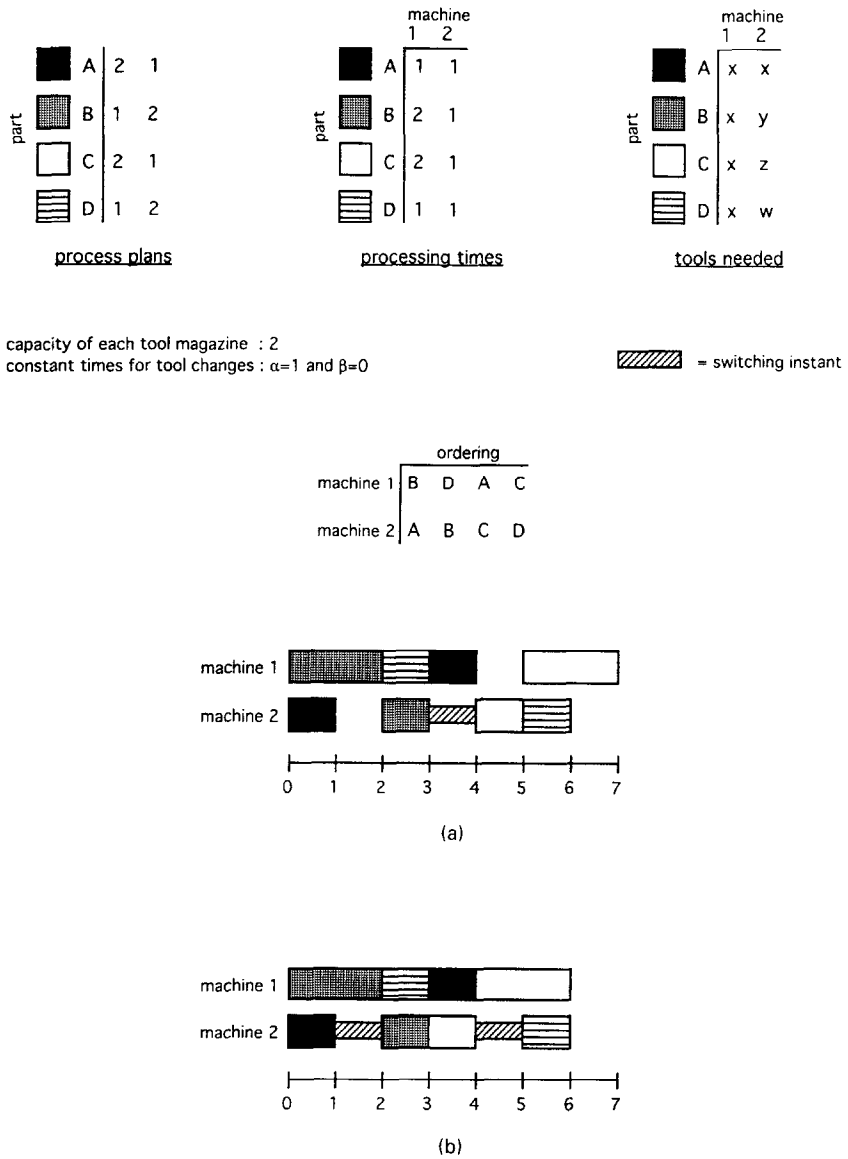
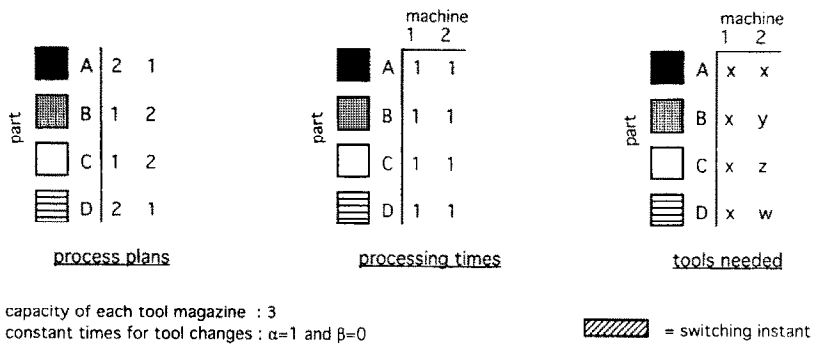
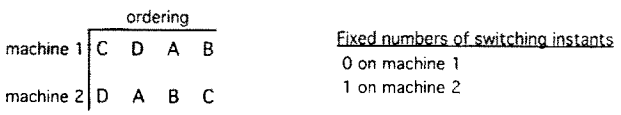


Fig. 2. (a) A schedule in 7 time units with one switching instant. (b) A schedule in 6 time units with two switching instants.

critical operation on part A, and then permute the two critical operations on parts B and C.

The next idea is to define $N(s)$ as the set of solutions which can be obtained from s by permuting two consecutive operations, without taking into account whether these two operations are critical or not. The main drawback of this definition is that





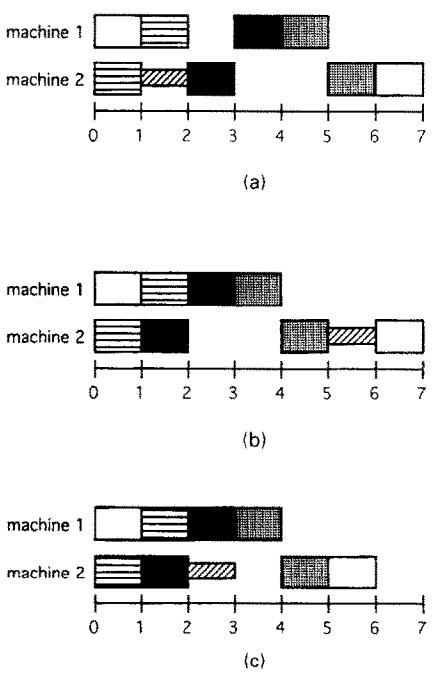


Fig. 3. (a) The switching instant is scheduled as early as possible. (b) The switching instant is scheduled as late as possible. (c) A better schedule in 6 time units.

most of the solutions in $N(s)$ have the same value as s since permuting two consecutive non-critical operations will usually not change the makespan; so, it is difficult to guide the search towards an optimal solution. We shall describe in the next section a neighborhood structure which is more suitable for the JPT.

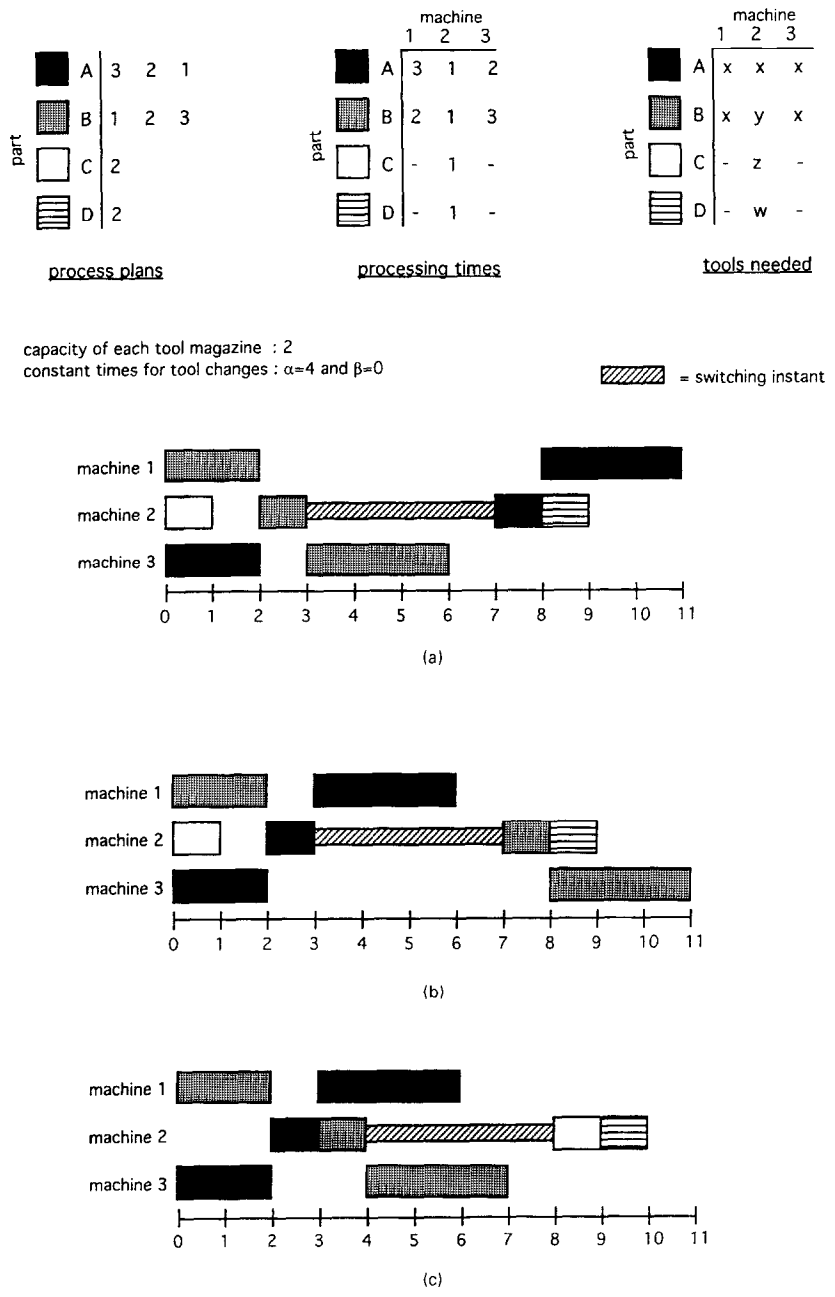


Fig. 4. (a) Solution s_1: the unique critical operations on machine 2 are the processing of parts A and B. (b) Solution s_2: the unique critical operations on machine 2 are the best processing of parts A and B. (c) Solution s_3: an optimal schedule.

4. The algorithm TOMATO

4.1. Scheduling the switching instants

As mentioned in Section 3.2, given an ordering of the operations, it is difficult to determine the best way for dealing with the tooling constraints. We have developed a three-phase heuristic method. In the first phase, all the machines are treated independently. We determine the set $\mathscr{P}$ of operations which immediately precede a switching instant. For a machine k, we denote:

- σ_k the number of switching instants on k;
- $\mathscr{M}_{k,q}$ $(1 \leqslant q \leqslant \sigma_k - 1)$ the set of tools needed to perform all operations on k which are between the qth and $(q+1)$th switching instants;
- $\mathscr{M}_{k,0}$ (resp. $\mathscr{M}_{k,\sigma_k}$) the set of tools needed to perform all operations on k which precede the first (resp. follow the last) switching instant.

The algorithm SWITCHING_INSTANT which is described below considers the ordered set $\{o_1, \ldots, o_r\}$ of operations on a machine k and sequentially schedules the switching instants only when forced to. Hence, if a switching instant precedes an operation o_i $(1 < i \leqslant r)$, this means that the set $\mathscr{T}(o_i)$ of tools needed to process o_i on k cannot be added to the tool magazine without exceeding its capacity. This is performed as follows.

Algorithm SWITCHING_INSTANT
let $\{o_1, \ldots, o_r\}$ be the ordered set of operations on a given machine k $(1 \leqslant k \leqslant m)$
set $i := 0$; $\mathscr{M}_{k,0} := \emptyset$; $\sigma_k := 0$;
$\mathscr{P} := \emptyset$ (* set of operations which immediately precede a switching instant *);
while $i < r$ **do**
  set $i := i + 1$;
  if $|\mathscr{M}_{k,\sigma_k} \cup \mathscr{T}(o_i)| > \mu_k$ **then**
schedule a switching instant immediately after operation o_{i-1};
    $\mathscr{P} := \mathscr{P} \cup \{o_{i-1}\}$;
    set $\sigma_k := \sigma_k + 1$; $\mathscr{M}_{k,\sigma_k} := \mathscr{T}(o_i)$;
  else set $\mathscr{M}_{k,\sigma_k} := \mathscr{M}_{k,\sigma_k} \cup \mathscr{T}(o_i)$;
end while.

It is clear that the number of switching instants determined by the above algorithm is minimal. In the following, we shall only consider schedules having that number of switching instants.

In the second phase of the proposed heuristic method, we determine the contents of the tool magazines. As in the first phase, all the machines are treated independently. Since tool replacements can only occur during the switching instants, a set of operations between two consecutive switching instants q and $(q+1)$ $(0 \leqslant q \leqslant \sigma_k)$ on a machine k can be considered as one unique operation which needs the set $\mathscr{M}_{k,q}$ of tools to be processed. Hence, given an ordered set of $\sigma_k + 1$ operations which

must be processed on machine k, we have to minimize the total number of tool replacements.

It has been proved by Tang and Denardo [28] that, given an ordered set of operations which must be performed on a unique machine, minimizing the total number of tool replacements is a polynomially solvable problem. Different proofs of this result are also given in [6, 24]. We solve the tool replacement problem by using the KTNS (keep tool needed soonest) policy proposed by Tang and Denardo [28]. This policy has the following properties:

- at any instant, no tool is inserted unless it is required by the next operation;
- if a tool must be inserted, the tools that are kept (not removed) are those needed the soonest.

For a machine k ($1 \leqslant k \leqslant m$), we denote:

- $\mathscr{C}_{k,q}$ ($1 \leqslant q \leqslant \sigma_k$) the contents of the tool magazine at the end of the qth switching instant;
- $\mathscr{C}_{k,0}$ the contents of the tool magazine at time zero.

For a tool τ, we define $\mathscr{L}_{k,q}(\tau)$ ($1 \leqslant k \leqslant m$; $1 \leqslant q \leqslant \sigma_k$) as the first instant at or after the qth switching instant on k, at which tool τ must appear in the tool magazine. If τ is not required by any operation following the qth switching instant, then we set $\mathscr{L}_{k,q}(\tau)$ equal to $\sigma_k + 1$. More formally:

$$\mathscr{L}_{k,q}(\tau) = \begin{cases} \sigma_k + 1 & \text{if } \tau \notin \bigcup_{r=q}^{\sigma_k} \mathscr{M}_{k,r}, \\ \min\{r \mid \tau \in \mathscr{M}_{k,r},\ q \leqslant r \leqslant \sigma_k\} & \text{otherwise.} \end{cases}$$

Hence, if $\mathscr{L}_{k,q}(\tau) = q$, then tool τ must belong to $\mathscr{C}_{k,q}$ since τ is needed to process at least one operation between the qth and the $(q+1)$th switching instant. Given a machine k ($1 \leqslant k \leqslant m$) with $\sigma_k > 0$, the tool replacement algorithm which follows the KTNS policy can now be described as follows.

Algorithm TOOL_REPLACEMENT

Put μ_k tools in $\mathscr{C}_{k,0}$ having minimal values of $\mathscr{L}_{k,0}(\tau)$;
(* i.e., those tools which are needed the soonest *)
set $q := 1$; (* switching instant counter *);
set $\mathscr{C}_{k,1} := \mathscr{C}_{k,0}$;
while $q < \sigma_k$ **do**
  if $\{\tau \mid \mathscr{L}_{k,q}(\tau) = q\} \subseteq \mathscr{C}_{k,q}$ **then**
    set $q := q + 1$;
    if $q \leqslant \sigma_k$ **then** set $\mathscr{C}_{k,q} := \mathscr{C}_{k,q-1}$;
  else
    choose any tool τ such that $\mathscr{L}_{k,q}(\tau) = q$ and $\tau \notin \mathscr{C}_{k,q}$;
    set $\mathscr{C}_{k,q} := \mathscr{C}_{k,q} \cup \{\tau\}$;
    remove a tool υ from $\mathscr{C}_{k,q}$ such that $\upsilon = \operatorname{argmax}_{\xi \in \mathscr{C}_{k,q}} \{\mathscr{L}_{k,q}(\xi)\}$
end while

Once the tool replacement problem has been solved on each machine, a feasible schedule can be obtained by solving a longest path problem. Indeed, all switching instants can be considered as additional operations whose duration is known, and the graph can then be constructed in the same way as described in Section 3.1 for the JP.

Given an ordering $\prec$ of the operations on the machines, a set $\mathscr{P}$ of operations which immediately precede a switching instant and a set $\mathscr{C} = \{\mathscr{C}_{k,q} | 1 \leqslant k \leqslant m; 0 \leqslant q \leqslant \sigma_k\}$ of contents of the tool magazines at time zero and at the end of the switching instants, we denote $\mathscr{S}(\prec, \mathscr{P}, \mathscr{C})$ the feasible schedule obtained by solving a longest path problem. For an operation i, we denote:

- $PM(i)$ the operation processed on M_i just before i;
- $SM(i)$ the operation processed on M_i just after i.

As mentioned above, the first two phases of the proposed heuristic method deal with all the machines independently. In a third phase, we treat all the machines simultaneously. We try to improve a solution by scheduling a switching instant on a machine when no part is ready to be processed on it.

Consider a feasible schedule $\mathscr{S}(\prec, \mathscr{P}, \mathscr{C})$ with a minimum number of switching instants and let i be any operation in $\mathscr{P}$. Assume that i precedes the qth switching instant on machine k and that $\mathscr{T}(i) \subseteq \mathscr{C}_{k,q}$. By moving the qth switching instant on k before operation i, we get a new set $\mathscr{P}' = (\mathscr{P} \setminus \{i\}) \cup PM(i)$ of operations which immediately precede a switching instant. Notice that by performing such a move, the contents of the tool magazine at the end of the qth switching instant on machine k need not be modified since all tools required to process operation i belong to $\mathscr{C}_{k,q}$. Hence, $\mathscr{S}(\prec, \mathscr{P}', \mathscr{C})$ is a schedule which satisfies the tooling constraints. Consider now the following times in $\mathscr{S}(\prec, \mathscr{P}, \mathscr{C})$:

- $t_1(k, q)$ is the completion time of operation $PM(i)$;
- $t_2(k, q)$ is the starting time of operation i;
- $t_3(k, q)$ is the starting time of operation $SP(i)$ if any, infinite otherwise;
- $t_4(k, q)$ is the completion time of the qth switching instant on k.

Note that $PM(i)$ always exists since otherwise, we could reduce the number of switching instants on k, which contradicts the fact that the number of switching instants in $\mathscr{S}(\prec, \mathscr{P}, \mathscr{C})$ is minimal.

Proposition. *Suppose* $t_3(k, q) \geqslant t_1(k, q) - t_2(k, q) + t_4(k, q)$. *Then, the makespan of* $\mathscr{S}(\prec, \mathscr{P}', \mathscr{C})$ *is smaller than or equal to the makespan of* $\mathscr{S}(\prec, \mathscr{P}, \mathscr{C})$.

Proof. In $\mathscr{S}(\prec, \mathscr{P}, \mathscr{C})$, the qth switching instant on machine k lasts $T_1 = t_4(k, q) - (t_2(k, q) + d_i)$ time units, while the idle time on k after the completion of $PM(i)$ lasts $T_2 = t_2(k, q) - t_1(k, q)$ time units.

Case 1: $T_1 \leqslant T_2$. In that case, we can construct a new schedule s by moving the qth switching instant on k before i and without performing any change in the starting time of any operation.

Case 2: $T_1 > T_2$. Consider the schedule s obtained from $\mathscr{S}(\prec, \mathscr{P}, \mathscr{C})$ by moving the qth switching instant on k before i. In that case, we must increase the starting time

of i by $T_1 - T_2$ time units. The new completion time of i will be $t_2(k, q) + d_i + (T_1 - T_2) = t_1(k, q) - t_2(k, q) + t_4(k, q)$. Now, since $t_1(k, q) \leqslant t_2(k, q)$, it follows that the operation $SM(i)$ on k (if any) need not be delayed. Moreover, since it is assumed that $t_3(k, q) \geqslant t_1(k, q) - t_2(k, q) + t_4(k, q)$, we do not have to change the starting time of operation $SP(i)$ (if any). This means that in s, the starting time of any operation $j \neq i$ can be set equal to the one in $\mathscr{S}(\prec, \mathscr{P}, \mathscr{C})$.

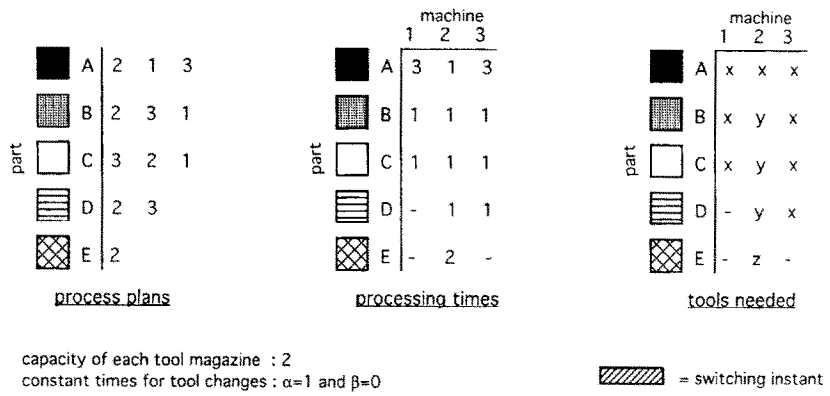
In each case, we have constructed a schedule s with a makespan smaller than or equal to the makespan of $\mathscr{S}(\prec, \mathscr{P}, \mathscr{C})$, and which satisfies the following properties:
- the operations in s are ordered according to $\prec$:
- the set of operations which immediately precede a switching instant is equal to $\mathscr{P}' = (\mathscr{P} \backslash \{i\}) \cup PM(i)$;
- the contents of the tool magazines in s are the same as those in $\mathscr{S}(\prec, \mathscr{P}, \mathscr{C})$.

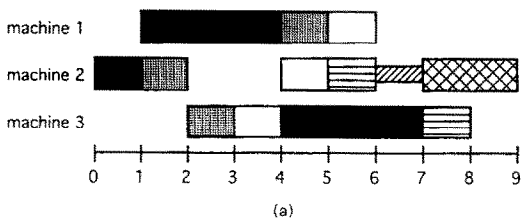
Among all schedules satisfying the above properties, $\mathscr{S}(\prec, P', \mathscr{C})$ has a minimum makespan since it is obtained by solving a longest path problem. Hence, the makespan of $\mathscr{S}(\prec, P', \mathscr{C})$ is smaller than or equal to the makespan of $\mathscr{S}(\prec, \mathscr{P}, \mathscr{C})$. □

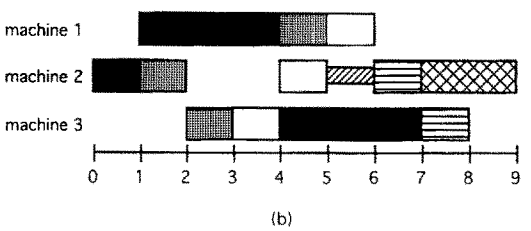
Notice that the makespan of $\mathscr{S}(\prec, \mathscr{P}', \mathscr{C})$ is possibly strictly smaller than the makespan of $\mathscr{S}(\prec, \mathscr{P}, \mathscr{C})$ since operation $SM(i)$ may perhaps be scheduled earlier. We have developed the following algorithm, which is based on the above proposition, for trying to improve the schedule of the switching instants.

Algorithm IMPROVE_SWITCHING_INSTANT
set $k := 1$ (* machine counter *);
set $q := 1$ (* switching instant counter *);
while $(k \leqslant m)$ **do**
  if $q > \sigma_k$ **then** set $k := k + 1$; $q := 1$
  else
    Let i be the operation which precedes the qth switching instant on machine k;
    if $\mathscr{T}(i) \subseteq \mathscr{C}_{k,q}$ **and** $t_3(k, q) \geqslant t_1(k, q) - t_2(k, q) + t_4(k, q)$ **then**
      move the qth switching instant on k between operations $PM(i)$ and i;
      solve a longest path problem (* construction of $\mathscr{S}(\prec, \mathscr{P}', \mathscr{C})$ *);
      set $k := 1$ and $q := 1$;
    else
      Set $q := q + 1$;
endwhile

As an example, consider the schedule represented in Fig. 5(a). There is a unique switching instant on machine 2 after the completion of part D. Assume that the contents $\mathscr{C}_{2,1}$ of the tool magazine of machine 2 after the switching instant is equal to $\{y, z\}$. We have $t_1(2, 1) = 5$, $t_2(2, 1) = 5$, $t_3(2, 1) = 7$ and $t_4(2, 1) = 7$. Since tool y needed to process part D on machine 2 belongs to $\mathscr{C}_{2,1}$ and since $t_3(2, 1) = t_1(2, 1) - t_2(2, 1) + t_4(2, 1) = 7$, the switching instant is moved before the processing of







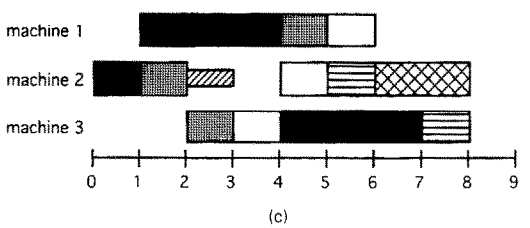


Fig. 5. (a) The switching instant has been scheduled as late as possible by the algorithm SWITCHING_INSTANT. (b) The switching instant has been moved before the processing of part D. (c) The switching instant has been moved before the processing of part C.

part D; one gets the schedule of Fig. 5(b). Now, $t_1(2, 1) = 2$, $t_2(2, 1) = 4$, $t_3(2, 1) = 5$, $t_4(2, 1) = 6$ and $t_3(2, 1) = 5 > 4 = t_1(2, 1) - t_2(2, 1) + t_4(2, 1)$. Since tool y needed to process part C on machine 2 belongs to $\mathscr{C}_{2,1}$, the switching instant is moved before the processing of part C and we obtain the schedule of Fig. 5(c). Finally, $t_3(2, 1) = 2 < 3 = t_1(2, 1) - t_2(2, 1) + t_4(2, 1)$; hence, the switching instant is not moved before the processing of part B. Notice that the schedule in Fig. 5(c) has a smaller makespan than those of Figs. 5(a) and (b).

Note that the use of heuristic methods to deal with the tooling constraints may constitute a hindrance to finding a global optimum for the JPT.

4.2. Other ingredients

The set X of feasible solutions is defined as the set of schedules $\mathscr{S}(\prec, \mathscr{P}, \mathscr{C})$ which satisfy the tooling constraints and which have a minimum number of switching instants, where

- $\prec$ is a feasible ordering of the operations;
- $\mathscr{P}$ is a set of operations which immediately precede a switching instant;
- $\mathscr{C} = \{\mathscr{C}_{k,q} \mid 1 \leqslant k \leqslant m;\ 0 \leqslant q \leqslant \sigma_k\}$ is a set of contents of the tool magazines at time zero and at the end of the switching instants.

Following Dell'Amico and Trubian's approach for the JP, we consider as neighbors of a solution s all solutions that can be obtained from s by moving a critical operation i in the first or in the last position of the block to which it belongs if the corresponding ordering is feasible; otherwise, operation i is moved in the position inside the block closest to the first or the last one and such that feasibility is preserved. As noticed in Section 3.2, it may be worthwhile to extend this neighborhood in the case of a JPT. We have decided to enlarge the definition of $N(s)$ by considering as neighbors of s those solutions obtained from s by moving the first (resp. last) operation i of a block before $PM(i)$ (resp. after $SM(i)$) if the corresponding ordering is feasible. These moves modify a critical path and this favors diversification of the search. In the experiments which are reported in Section 5, we have observed that moves to this new type of neighbors are performed in about 15% of the cases. They are beneficial since the results obtained with the proposed extended neighborhood have proven to be systematically better than those obtained by using Dell'Amico and Trubian's neighborhood structure.

We have tested a variant of this neighborhood structure where consecutive operations which are non-critical are allowed to be permuted. We got results which were systematically worse than those reported in Section 5 and we have observed that about 60% of the moves involved a non-critical operation. This can easily be explained by the fact that many solutions of this modified neighborhood $N(s)$ have the same value as s. In this case, the algorithm will move a non-critical operation without increasing the value of the objective function instead of moving a critical operation which would lengthen the longest path. It is therefore difficult to escape a local optimum.

Let s' be a neighbor of a solution s obtained by moving an operation on a machine k. To evaluate s', we first determine the operations on k which immediately precede a switching instant by using the algorithm SWITCHING_INSTANT. We then apply the algorithm TOOL_REPLACEMENT on k and compute the makespan of s' by solving a longest path problem (see Section 3.2). Once it has been decided to move from the current solution s to a neighbor solution $s' \in N(s)$, we try to improve the schedule of the switching instants in s' by using the algorithm IMPROVE_SWITCHING_INSTANT.

In order to get a better evaluation of the neighbors of a solution s, we have tried to apply the procedure IMPROVE_SWITCHING_INSTANT to each of them. An important increase of the computational effort has been observed which can easily be explained by the following fact. Since the procedure SWITCHING_INSTANT schedules all switching instants only when forced to, it often happens that many of them can be scheduled earlier by using the improvement procedure. At each change of the schedule of a switching instant, a longest path problem is solved, and the time needed to detect such a possible change is, in the worst case, proportional to the number of switching instants. We have therefore decided to use the procedure IMPROVE_SWITCHING_INSTANT only once per iteration.

Notice that the proposed neighborhood structure does not necessarily satisfy property $\mathfrak{P}$ defined in Section 2. Indeed, for the problem represented in Fig. 6, the state space graph $\mathcal{G}$ contains two connected components.

The tabu list T is a set of operations which are not allowed to be moved for a certain number of iterations. When an operation i is moved after or before an operation j on a machine k, operation i is introduced in the tabu list. Furthermore, if j is an immediate predecessor or successor of i, then operation j is also added to the tabu list: indeed, in that case, s could be obtained from s' by moving j after or before i. The length of the tabu list is chosen at each iteration at random in the interval $[n, \lfloor 3n/2 \rfloor]$, according to a uniform distribution.

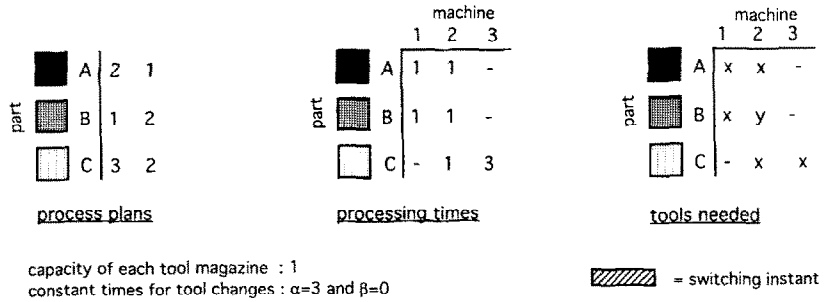
The objective function f is the makespan. For a solution s and a machine k, let $E_k(s)$ denote the time at which all operations are completed on k. We have

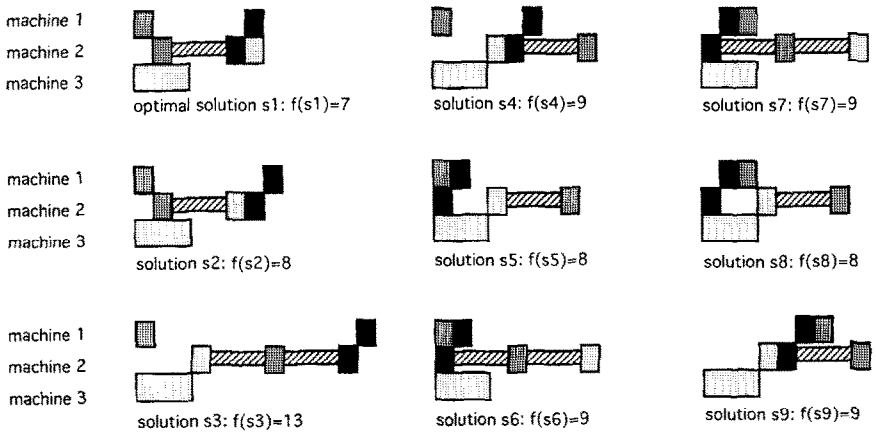
$$f(s) = \max_{1 \leq k \leq m} \{E_k(s)\}.$$

As mentioned in Section 2, for moving from a solution s to a neighbor solution s' in $N(s)$, we first have to generate a subset $V^*(s)$ of $N(s)$. We then have to choose a best solution s' in $V^*(s)$ according to a function g which may be different from f. Usually, many neighbors of a solution s have the same makespan as s. We have therefore defined the two following discriminating functions h_1 and h_2:

- $h_1(s)$ is the total duration of the switching instants in s;
- $h_2(s) = \sum_{k=1}^{m} E_k(s)^2$.

By minimizing $h_2(s)$, we try to avoid having too many machines whose completion time is close to the makespan.





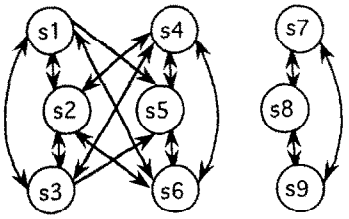


Fig. 6.

Let F be a weighted combination of f and h_1:

$$F(s) = w_1 f(s) + w_2 h_1(s),$$

where w_1 and w_2 are positive parameters which give relative weights to the two components of F. Moreover, let *f_best* (resp. *F_best*) denote the value of the best solution according to f (resp. F) encountered so far during the resolution.

For choosing a solution in $N(s)$, we first determine the subset $N'(s)$ of neighbors s' which are either not tabu or such that $(f(s') < f\_best)$ or $(F(s') < F\_best)$. Denote $V^*(s)$ the set containing the best neighbors in $N'(s)$ according to the function F:

$$V^*(s) = \left\{ s' \mid s' = \operatorname*{argmin}_{x \in N'(s)} \{F(x)\} \right\}.$$

We move from s to a solution s' in $V^*(s)$ for which h_2 is minimal:

$$s' = \operatorname*{argmin}_{x \in V^*(s)} \{h_2(x)\}.$$

Finally, the iterative process is stopped as soon as *max_iteration* iterations have been performed without improving the values of *f_best* or *F_best*.

4.3. The algorithm

The proposed heuristic method for solving the JPT is named TOMATO (for TOol MAnagement with Tabu Optimization) and can be summarized as follows:

Algorithm TOMATO
Initialization
find an initial solution for the JP by using Taillard's algorithm [27];
let s be the schedule obtained by applying the algorithms SWITCHING_INSTANT and TOOL_REPLACEMENT on each machine and by solving a longest path problem;
improve s using the algorithm IMPROVE_SWITCHING_INSTANT;
set $s^* := s$ (* best solution according to f *);
set $f\_best := f(s)$; $F\_best := F(s)$ (* best values of f and F *);
set *iteration_best* := 0 (* last iteration when *f_best* or *F_best* has been modified *);
set *iteration_counter* := 0 (* iteration counter *);
set $T := \emptyset$ (* the tabu list is initially empty *);
while $(iteration\_counter - iteration\_best) \leqslant max\_iteration$ **do**
set *iteration_counter* := *iteration_counter* + 1;
set *tabu_length* := random number $\in [n, \lfloor 3n/2 \rfloor]$;
evaluate each solution s' of $N(s)$ by using the algorithms SWITCHING_INSTANT and TOOL-REPLACEMENT for the machine on which an operation is moved and by solving a longest path problem;
determine the subset $N'(s) \subseteq N(s)$ of neighbors s' of s which are either not tabu or such that $((f(s') < f\_best)$ or $(F(s') < F\_best))$;
determine the subset $V^*(s) \subseteq N'(s)$ containing the best neighbors of s according to F;
choose the best solution s' in $V^*(s)$ according to the discriminating function h_2;
improve s' using the algorithm IMPROVE_SWITCHING_INSTANT;
set $s := s'$ and update s^*, *f_best*, *F_best*, *iteration_best* and T;
end while

The solution s^* is the best schedule found by TOMATO and *f_best* is its makespan. We evaluate the efficiency of this algorithm in the next section.

5. Computational results

Benchmark problems have been generated by Widmer [31] who proposed a first adaptation of tabu search to the JPT. We have considered the problems of type A, B, E, F, G, H, I and J described in [31] (the problems of type C and D are the same as those of type B with additional constraints such as due dates). These problems have a number of machines and a number of parts varying from 1 to 10. They constitute a good base for evaluating the algorithm TOMATO.

We have generated 10 problems of each type. All problems are exactly the same instances as those in [31]. In these problems, the parts have to visit every machine exactly once and the process plans have been randomly generated. For each operation, the processing time is an integer which is randomly generated and uniformly distributed over the interval [1, 5], and the number of required tools is an integer which is randomly generated and uniformly distributed over the interval [1, 10]. The capacity of the tool magazines is equal to 10. The duration of a switching instant does not depend on the number of tools which are changed and is equal to one time unit; hence $\alpha = 1$ and $\beta = 0$.

We have set $w_1 = 2$ and $w_2 = 1$ and all the tests have been performed on a Silicon Graphics workstation. Each problem has been run 10 times; this gives a total amount of 800 experiments. Table 1 contains the summary of the results and a comparison between the algorithm TOMATO and the adaptation of tabu search for the JPT which was proposed by Widmer [31] and which is named JEST. For each of the 80 problems, we define the *reference objective function* as the best value $f(s^*)$ of s^* among the 10 runs of TOMATO and JEST, with *max_iteration* set equal to 2000.

For each problem type, we report the results obtained with given values of the parameter *max_iteration*. Recall that *max_iteration* is a limit on the number of successive iterations without improvement in the best solution according to f or F.

The column *%_best* contains the percentage of problems for which the *reference objective function* has been reached. For each one of the 80 problems P, let $M(P)$ and $W(P)$ denote, respectively, the mean and the worst value of the objective function over the 10 runs. For a problem type $\wp$, the values in the columns *mean_δ* and *worst_δ* represent the average and the worst percentage above the *reference objective function*; these are computed as follows:

$$mean\_\delta(\wp) := 100 \sum_{P \in \wp} \frac{M(P) - \textit{reference objective function}}{\textit{reference objective function}},$$

$$worst\_\delta(\wp) := 100 \sum_{P \in \wp} \frac{W(P) - \textit{reference objective function}}{\textit{reference objective function}}.$$

Table 1
Results obtained with TOMATO and JEST for problems of type A, B, E, F, G, H, I and J from [31]

Problem type	Size ($n \times m$)	*max iteration*	*%_best*		*mean δ*		*worst δ*		*mean_iteration*		CPU time (s)	
			TOMATO	JEST	TOMATO	JEST	TOMATO	JEST	TOMATO	JEST	TOMATO	JEST
A	10×1	0	89	58	0.32	1.45	0.58	2.44	2.96	1.46	0.00	0.00
		100	100	100	0.00	0.00	0.00	0.00	103.40	124.39	0.07	0.30
B	3×3	0	98	3	0.10	55.34	0.48	61.29	1.78	1.23	0.00	0.00
		100	100	96	0.00	0.54	0.00	3.93	101.85	137.95	0.05	0.12
		200	100	99	0.00	0.13	0.00	1.25	201.85	243.87	0.10	0.21
		300	100	100	0.00	0.00	0.00	0.00	301.85	345.98	0.15	0.29
E	3×4	0	99	0	0.05	61.73	0.50	67.93	3.45	1.18	0.00	0.00
		100	100	71	0.00	2.28	0.00	10.79	103.50	152.83	0.09	0.16
		500	100	88	0.00	0.81	0.00	1.19	503.50	616.38	0.42	0.63
		1000	100	89	0.00	0.76	0.00	1.19	1003.50	1125.77	0.84	1.15
		2000	100	89	0.00	0.76	0.00	1.19	2003.50	2125.77	1.68	2.16
F	3×5	0	93	0	0.29	69.29	0.42	71.07	25.27	1.11	0.03	0.00
		100	99	74	0.04	1.62	0.42	6.13	126.60	170.10	0.14	0.25
		200	100	82	0.00	1.05	0.00	3.97	227.85	288.43	0.26	0.41
		500	100	92	0.00	0.35	0.00	1.45	527.85	657.76	0.60	0.94
		1000	100	97	0.00	0.12	0.00	0.42	1027.85	1225.73	1.17	1.75
		2000	100	98	0.00	0.08	0.00	0.42	2027.85	2240.02	2.32	3.22
G	8×4	0	78	0	0.89	130.11	3.55	133.18	99.22	1.10	0.31	0.01
		100	93	18	0.26	9.07	0.57	19.54	214.85	248.54	0.68	3.38
		500	96	38	0.11	4.13	0.29	9.13	722.32	949.55	2.31	12.82
		1000	98	52	0.06	2.52	0.29	5.68	1358.16	1800.67	4.35	24.26
		2000	98	57	0.06	1.74	0.29	3.22	2490.65	3296.05	7.97	44.53
H	5×5	0	47	0	2.53	108.38	4.21	111.27	43.10	1.17	0.21	0.01
		100	73	8	1.09	11.44	3.11	21.99	164.13	232.22	0.49	1.17
		500	87	29	0.45	5.05	1.09	10.09	633.68	923.61	1.55	4.70
		1000	89	41	0.38	3.70	0.70	8.00	1167.04	1644.67	2.77	8.36
		2000	97	52	0.10	2.63	0.70	6.24	2294.38	3057.25	5.35	15.44
I	5×10	0	59	0	1.37	182.04	2.32	183.55	51.50	1.08	0.57	0.01
		100	88	0	0.43	32.73	1.14	56.51	167.13	317.57	1.90	4.54
		500	91	1	0.32	13.41	0.91	23.59	600.48	1132.05	6.86	16.22
		1000	92	3	0.27	9.59	0.91	18.03	1127.74	2060.58	12.95	29.51
		2000	92	8	0.27	7.58	0.91	13.99	2141.50	3671.23	24.67	52.43
J	10×10	0	6	0	4.97	331.51	8.04	332.17	25.12	1.05	0.64	0.13
		100	12	0	3.40	69.48	5.57	109.39	164.14	535.82	4.25	64.55
		500	22	0	2.33	28.58	3.54	47.92	764.18	1800.85	19.94	216.73
		1000	30	0	1.77	22.81	3.52	33.00	1564.77	2871.04	40.97	345.84
		2000	44	0	1.28	19.21	2.79	27.30	3171.36	4798.90	83.00	576.95

The column *mean_iteration* contains the mean number of iterations which have been performed. Finally, the CPU times are given in seconds.

Notice that the objective function used in [31] is not the makespan. This explains why the values of *%_best* for JEST given in Table 1 are sometimes larger than those reported in [31].

It appears clearly that TOMATO is more efficient than JEST. For the problems of type A and B, when *max_iteration* is large enough (that is greater or equal to 300), both algorithms find the best known solutions in all cases. The values obtained with *max_iteration* = 0 correspond to the values of the first local optimum: we observe that for these two problem types, the first local optimum obtained with TOMATO is equal to the best known solution in about all cases. This is not the case for JEST since only 58% (resp. 3%) of the first local optima are equal to s^* for problems of type A (resp. B).

For the problems of type E, F, G and I, we notice that TOMATO finds the best known solutions in more than 90% of the cases. The algorithm JEST is much less robust than TOMATO since the measure *%_best* is equal to about 50% for the problems of type G and H, and to 8% for the problems of type I.

The algorithm JEST has not found any best known solution for the problems of type J while TOMATO succeeds in minimizing the makespan in 44% of the cases.

The robustness of TOMATO appears also clearly in the columns *mean_δ* and *worst_δ*. For the problems of type J, TOMATO finds the best known solution in less than 50%, but we observe that the mean (resp. worst) values of s^* are 1.3% (resp. 2.8%) above the *reference objective function*. These mean and worst values are even less than 1% above the *reference objective function* for the problems of type A, B, E, F, G, H and I. For comparison, JEST finds solutions with *worst_δ* = 27.3% and *mean_δ* = 19.2% for the problems of type J.

As mentioned above, all tests concerning the algorithms TOMATO and JEST have been performed on a Silicon Graphics workstation. This is the reason why the CPU times for the algorithm JEST do not correspond to those given in [31]. It appears clearly that JEST is a much slower algorithm than TOMATO. Indeed, for the problems of type J, JEST needs about 0.12 s per iteration while TOMATO performs one iteration in about 0.025 s. This is due to the fact that the neighborhood used in JEST is much larger than the one of TOMATO. In JEST, a feasible solution s' is defined as a neighbor of s if it is obtained from s by moving an operation i to another position on M_i. This induces a neighborhood of size $O(mn^2)$. In the worst case, the neighborhood used in TOMATO is of size $O(mn)$.

Moreover, for the problems of type E, G, H, I and J, the results obtained by using JEST with *max_iteration* = 2000 are worse than those obtained with TOMATO and *max_iteration* = 0. As an example, for the problems of type J, JEST provides in about 10 min a solution with a makespan which is in average 19% above the best known makespan. The algorithm TOMATO needs about half a second for finding a solution whose value is in average (resp. in the worst case) 5% (resp. 8%) above the *reference objective function*.

Benchmark problems for the JP have been generated by Fisher and Thompson [10], Lawrence [21] and Adams et al. [1]. We have incorporated tooling constraints into 45 of these problems: 40 from Lawrence (LA1-40) and 5 from Adams et al. (ABZ5-9). This has been done in the following way which can easily be reproduced by any researcher interested in testing other heuristic solution methods for the JPT.

We have first assumed that a tool is never required by two different operations. Hence, $\mathcal{T}(i) \cap \mathcal{T}(j) = \emptyset$ for each pair i, j of operations. Let T denote the mean processing time of an operation:

$$\mathsf{T} = \sum_{i=1}^{N} d_i / N.$$

Let $\{o_1, \ldots, o_r\}$ be the ordered set of operations in the process plan of a part p $(1 \leqslant p \leqslant n)$. The number of tools required by operation o_i $(1 \leqslant i \leqslant r)$ has been set equal to i. The capacity of the tool magazines has been set equal to m (the number of machines) and we assume that $\alpha = \mathsf{T}/10$ and $\beta = 0$.

These 45 problems can be considered as a new set of benchmark problems for the JPT. We have tested the algorithm TOMATO by setting $w_1 = 2$ and $w_2 = 1$. Each problem has been run 10 times and the results are summarized in Table 2.

For each problem, the *reference objective function* represents the best value $f(s^*)$ of s^* among the 10 runs of TOMATO with *max_iteration* = 2000. As in Table 1, *%_best* corresponds to the percentage of experiments for which the *reference objective function* has been reached. *Mean_δ* and *worst_δ* represent the average and the worst percentage above the *reference objective function* over the 10 runs. The column *initial solution* provides an evaluation of the solutions obtained by only performing the initialization phase of the algorithm TOMATO, while the column *final solution* gives information about the final solutions s^* obtained by setting *max_iteration* = 2000. The last line of Table 2 contains a summary of the results for the 45 problems.

It can be observed that also in this case, TOMATO appears to be very robust since the mean and the worst values of s^* are in average less than 1% above the *reference objective function*.

6. Final remarks and conclusions

Only few models have been proposed in the literature for solving job shop problems with tooling constraints. The algorithm TOMATO which is described in this paper is robust and provides in a short computational time solutions of much better quality than those reported in [31]. In our opinion, TOMATO should only be applied to job shop problems in which the tool magazines are of relatively limited capacity: this happens in small and medium industry. When the capacity of the tool magazines increases and becomes much larger than the average number of tools needed by an operation, the problem becomes more or less similar to the JP; in this case, a solution of the JP could be obtained with the tabu search adaptation proposed by Dell'Amico

Table 2
Results obtained with TOMATO for problems LA1-40 [21] and ABZ5-9 [1]

Problem	Size ($n \times m$)	Initial solution			Final solution			*Reference objective function*
		%_best	*mean_δ*	*worst_δ*	*%_best*	*mean_δ*	*worst_δ*	
LA1	10 × 5	10	0.61	0.74	30	0.50	0.74	676
LA2	10 × 5	100	0.00	0.00	100	0.00	0.00	675
LA3	10 × 5	40	0.30	0.49	60	0.20	0.49	609
LA4	10 × 5	0	1.68	2.37	50	1.02	2.37	590
LA5	10 × 5	0	0.67	0.67	100	0.00	0.00	601
LA6	15 × 5	90	0.05	0.53	100	0.00	0.00	946
LA7	15 × 5	20	0.29	0.89	100	0.00	0.00	902
LA8	15 × 5	50	0.34	1.13	100	0.00	0.00	883
LA9	15 × 5	100	0.00	0.00	100	0.00	0.00	971
LA10	15 × 5	70	0.11	0.72	100	0.00	0.00	978
LA11	20 × 5	60	0.16	0.80	100	0.00	0.00	1247
LA12	20 × 5	70	0.13	0.76	100	0.00	0.00	1059
LA13	20 × 5	90	0.02	0.17	100	0.00	0.00	1180
LA14	20 × 5	0	0.38	0.38	100	0.00	0.00	1317
LA15	20 × 5	20	0.63	1.22	90	0.04	0.41	1227
LA16	10 × 10	0	1.25	2.70	30	0.71	2.18	963
LA17	10 × 10	0	2.50	2.65	10	1.60	2.40	793
LA18	10 × 10	0	1.28	3.08	60	0.19	0.80	876
LA19	10 × 10	20	0.72	1.72	20	0.69	1.38	870
LA20	10 × 10	0	0.52	0.97	10	0.19	0.22	925
LA21	15 × 10	10	1.29	2.19	10	0.88	2.01	1097
LA22	15 × 10	60	0.21	1.03	80	0.05	0.41	971
LA23	15 × 10	20	0.54	1.21	100	0.00	0.00	1072
LA24	15 × 10	0	1.56	2.35	10	1.12	2.15	979
LA25	15 × 10	0	1.61	2.25	10	1.17	1.86	1022
LA26	20 × 10	0	0.77	1.56	90	0.06	0.63	1278
LA27	20 × 10	0	1.92	2.85	10	1.25	2.15	1300
LA28	20 × 10	20	0.51	1.48	30	0.27	0.70	1281
LA29	20 × 10	0	2.02	3.93	10	1.48	2.86	1222
LA30	20 × 10	40	0.31	0.63	90	0.04	0.35	1420
LA31	30 × 10	0	0.59	1.07	100	0.00	0.00	1874
LA32	30 × 10	0	0.68	0.88	30	0.18	0.26	1935
LA33	30 × 10	0	0.58	1.18	90	0.02	0.22	1787
LA34	30 × 10	0	0.99	1.71	20	0.25	0.55	1811
LA35	30 × 10	0	1.20	1.68	10	0.30	0.51	1968
LA36	15 × 15	10	0.82	1.38	50	0.24	0.76	1308
LA37	15 × 15	20	0.75	2.04	30	0.46	1.15	1473
LA38	15 × 15	0	1.38	2.60	10	1.07	2.27	1233
LA39	15 × 15	0	1.47	2.35	10	1.11	1.57	1275
LA40	15 × 15	0	1.11	1.63	10	0.82	1.16	1288
ABZ5	10 × 10	60	0.14	0.47	60	0.10	0.47	1271
ABZ6	10 × 10	100	0.00	0.00	100	0.00	0.00	970
ABZ7	20 × 15	0	1.39	2.46	10	0.95	1.79	691
ABZ8	20 × 15	0	2.05	2.71	10	1.57	2.57	701
ABZ9	30 × 15	0	1.76	4.04	10	1.27	2.79	717
Average results		24.0	0.82	1.50	54.4	0.43	0.89	

and Trubian, and the few switching instants can then be scheduled by using the algorithms SWITCHING_INSTANT, TOOL_REPLACEMENT and IMPROVE_SWITCHING_INSTANT.

TOMATO seems to be a very robust algorithm since we have observed on benchmark problems that it produces in less than 1 min solutions whose mean value is at most 1% above the best known values. For comparisons, a previous work on some of these benchmark problems reports results which are obtained in a few minutes and which are in some cases about 20% above the solution values found by TOMATO.

TOMATO is a flexible method which can be used by a production manager who wants either a quick but not precise estimation of the makespan or an accurate solution. The desired trade-off between computational time and solution quality can be established by adjusting the parameter *max_iteration*. TOMATO can be used as a descent method that stops as soon as a local optimum has been reached: in the experiments which are reported in Section 5, we have observed that TOMATO provides in less than 1 second solutions whose value is in the worst case less than 8% above the best known solution values. By using TOMATO during 1 min, we have noticed that this percentage above the best known solution values is decreased to 3%.

The complexity of the following problem is still open: given the process plans of the part types and an ordering of the operations on the machines, determine a schedule of the operations and of the switching instants which minimizes the makespan. We have developed a three-phase heuristic method for solving this problem. One possible way for improving the procedure IMPROVE_SWITCHING_INSTANT would consist in defining a measure of criticality of every machine and optimizing more critical machines first. It is not difficult to build examples where the three-phase heuristic method does not provide an optimal schedule of the switching instants. This constitutes of course a hindrance to finding a global optimum for the JPT. Future work will consist in trying to develop a better heuristic method or an exact polynomial algorithm (if any).

The neighborhood structure used in TOMATO does not induce a state space graph $\mathcal{G}$ having property $\mathfrak{P}$. Hence, there exist initial solutions for which TOMATO will never find a global optimum. This drawback could be reduced by adding a multiple restart strategy to TOMATO or by enlarging the neighborhood structure. For example, one could define $N(s)$ as the set of solutions which can be obtained from s by permuting two consecutive operations, without taking into account whether these two operations are critical or not. It should however be noticed that it would be difficult to escape a local optimum with such a neighborhood structure since many neighbors of a solution s would have the same makespan as s and would be preferred to those which increase the makespan. We have also observed that an increase of the size of the neighborhood involves an increase of the time needed to perform one iteration: this explained the slowness of JEST when compared to TOMATO.

Future development in this topic will consist in defining a neighborhood structure of reasonable size which favors the escape from local optima and which induces a state space graph having property $\mathfrak{P}$.

Acknowledgements

The authors would like to thank Eric Taillard who provided them with the code for the job shop problem without tooling constraints.

They also thank the referees for constructive remarks on an earlier version of this paper.

References

[1] J. Adams, E. Balas and D. Zawack, The shifting bottleneck procedure for job shop scheduling, Management Sci. 34 (1988) 391–401.
[2] K.R. Baker, Introduction to Sequencing and Scheduling (Wiley, New York, 1974).
[3] J.F. Bard, A heuristic for minimizing the number of tool switches on a flexible machine, IIE Trans. 20 (1988) 382–391.
[4] J.W. Barnes and M. Laguna, A tabu search experience in production scheduling, Ann. Oper. Res. 41 (1993) 141–156.
[5] M. Berrada and K.E. Stecke, A branch and bound approach for machine load balancing in flexible manufacturing systems, Management Sci. 32 (1986) 1316–1335.
[6] Y. Crama, A.W.J. Kolen, A.G. Oerlemans and F.C.R. Spieksma, Minimizing the number of tool switches on a flexible machine, Internat. J. Flexible Manufacturing Systems 6 (1994) 33–54.
[7] M. Dell'Amico and M. Trubian, Applying tabu search to the job-shop scheduling problem, Ann. Oper. Res. 41 (1993) 231–252.
[8] D. de Werra and A. Hertz, Tabu search techniques: a tutorial and an application to neural networks, OR Spektum 11 (1989) 131–141.
[9] D. de Werra and M. Widmer, Loading problems with tool management in flexible manufacturing systems: a few integer programming models, Internat. J. Flexible Manufacturing Systems 3 (1990) 71–82.
[10] S. Fisher and G.L. Thompson, Probabilistic learning combinations of local job shop scheduling rules, in: J.F. Muth and G.L. Thompson, eds., Industrial Scheduling (Prentice-Hall, Englewood Cliffs, NJ, 1963) 225–251.
[11] J.P. Follonier, On grouping parts and tools and balancing the workload on a flexible manufacturing system, Working paper ORWP 92/05, DMA-EPFL, Lausanne (1992).
[12] J.P. Follonier, Minimization of the number of tool switches on a flexible manufacturing machine, Working Paper ORWP 92/13, DMA-EPFL, Lausanne (1992).
[13] F. Glover, Future paths for integer programming and links to artificial intelligence, Comput. Oper. Res. 13 (1986) 533–549.
[14] F. Glover, Tabu search (part I), ORSA J. Comput. 1 (1989) 190–206.
[15] F. Glover, Tabu search (part II), ORSA J. Comput. 2 (1990) 4–32.
[16] F. Glover, E. Taillard and D. de Werra, A user's guide to tabu search, Ann. Oper. Res. 41 (1993) 3–28.
[17] P. Hansen, The steepest ascent mildest descent heuristic for combinatorial programming, presented at the congress on Numerical Methods in Combinatorial Optimization, Capri (1986).
[18] A. Hertz and D. de Werra, The tabu search metaheuristic: how we used it, Ann. Math. Artificial Intelligence 1 (1990) 111–121.
[19] A. Hertz, E. Taillard and D. de Werra, Tabu search, in: J.K. Lenstra, ed., Local Search in Combinatorial Optimization, Working Paper ORWP 92/18, DMA-EPFL, Lausanne (1992).
[20] C.P. Koulamas, Total tool requirements in multi-level machining systems, Internat. J. Prod. Res. 29 (1991) 417–437.
[21] S. Lawrence, Supplement to "Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques", GSIA, Carnegie-Mellon University (1984).
[22] Y. Mottet and M. Widmer, Dynamic scheduling and tool loading, in: V.C. Venkatesh and J.A. McGeough, eds., Computer-Aided Production Engineering (Elsevier, Amsterdam, 1991) 325–332.

[23] E. Nowicki and C. Smutnicki, A fast taboo search algorithm for the job shop problem, Preprint 93/8, Instytut Cybernetyka Technicznej, Politechniki Wroclawskiej, Wroclaw (1993).
[24] C. Privault, Modèles mathématiques pour la gestion off-line et on-line des changements d'outils sur une machine flexible, Thèse de doctorat, Université Joseph Fourier, Grenoble (1994).
[25] C. Proust, J.N.D. Gupta and V. Deschamps, Flowshop scheduling with set-up, processing and removal times separated, Internat. J. Prod. Res. 29 (1991) 479–493.
[26] B. Roy and B. Sussmann, Les problèmes d'ordonnancement avec contraintes disjonctives, Note DS n.9 bis, SEMA, Montrouge (1964).
[27] E. Taillard, Parallel taboo search technique for the jobshop scheduling problem, ORSA J. Comput. 6 (1994) 108–117.
[28] C.S. Tang and V. Denardo, Models arising from a flexible manufacturing machine: Part I: Minimization of the number of tool switches, Part II: Minimization of the number of switching instants, Oper. Res. 36 (1988) 767–784.
[29] P.J.M. van Laarhoven, E.H.L. Aarts and J.K. Lenstra, Job shop scheduling by simulated annealing, Oper. Res. 40 (1992) 113–125.
[30] D. Veeramani, D.M. Upton and M.M. Barash, Cutting-tool management in computer-integrated manufacturing, Internat. J. Flexible Manufacturing Systems 3/4 (1992) 237–265.
[31] M. Widmer, Job shop scheduling with tooling constraints: a tabu search approach, J. Oper. Res. Soc. 42 (1991) 75–82.
[32] M. Widmer and A. Hertz, A new heuristic method for the flow shop sequencing problem, European J. Oper. Res. 41 (1989) 186–193.